Preprint
UCRL-JC-141942

PADRE: A Parallel Asynchronous Data Routing Environment

B. Gunney, D. Quinlan

This article was submitted to
Joint Association for Computing Machinery Java Grande-International Scientific Computing in Object-Oriented Parallel Environments Conference, Palo Alto, CA, June 2-4, 2001

January 8, 2001

This work was performed under the auspices of the United States Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

PADRE: A Parallel Asynchronous Data Routing Environment *

Brian Gunney

Dan Quinlan
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
gunneyb,dquinlan@llnl.gov

Abstract

Increasingly in industry, software design and implementation is object-oriented, developed in C++ or Java, and relies heavily on pre-existing software libraries (e.g. the Microsoft Foundation Classes for C++, the Java API for Java). A similar but more tentative trend is developing in high-performance parallel scientific computing. The transition from serial to parallel application development considerably increases the need for library support: task creation and management, data distribution and dynamic re-distribution, and inter-process and inter-processor communication and synchronization must be supported.

PADRE is a library to support the interoperability of parallel applications. We feel there is significant need for just such a tool to compliment the many domain-specific application frameworks presently available today, but which are generally not interoperable.

1 Introduction

Increasingly in industry, software design and implementation is object-oriented, developed in C++ or Java, and relies heavily on pre-existing software libraries (e.g. the Microsoft Foundation Classes for C++, the Java API for Java). A similar but more tentative trend is developing in high-performance parallel scientific computing. The transition from serial to parallel application development considerably increases the need for library support: task creation and management, data distribution and dynamic re-distribution, and inter-process and inter-processor communication and synchronization must be supported.

In large-scale scientific computing these issues have tended to be addressed in an ad-hoc, per-application basis with relatively primitive support libraries, e.g. the MPI, PVM, and PTHREADS libraries. For the management of distributed data there does not exist even consensus-standard libraries or APIs. Efforts to provide specialized high-level libraries and compiler support have been fragmented and disjoint and so have not integrated well. As a consequence they have not given rise to a publicly-available cohesive software infrastructure that significantly facilitates the development of large-scale complex parallel

*This work is funded (in part?) by the Department of Energy's Division of Mathematical, Information, and Computational Sciences under contract number ???.

scientific applications. Again in the particular case of distributed data management, existing libraries tend to be force-fit to applications rather than the applications dictating the characteristic of the data management component.

The DOE 2000 Scientific Template Library (SciTL) seeks to bring some order to the chaos by providing a comprehensive infrastructure for the creation of the specialized parallel libraries needed for specific application areas. The emphasis is on the support of parallel frameworks presently in use for both Accelerated Strategic Computing Initiative (ASCI) and Energy Research (ER) projects within DOE, but SciTL will be publicly available and is expected to form a basis for work elsewhere. SciTL comprises four principal libraries. These libraries provide mutually disjoint functionality, and though they are nominally intended to be used together, an express goal is that there be no strict interdependence—each may be used on its own. One of these libraries is PADRE, a library for the management of distributed data.

The lack of any general-purpose distributed-data management facilities (henceforth *data distribution libraries* motivated the development of libraries such as Multiblock PARTI [?], PGSLib [?, ?], and KeLP [?, ?]. At this level of abstraction (unlike at the PVM/MPI level) it appears unreasonable to hope for a single satisfactory solution because of widely differing distribution and communication requirements, sometimes even within a single application or parallel library.

The sophistication of distributed-data management libraries (or *distribution libraries*) is such that it behooves the parallel application or library designer to use existing libraries rather than build them from scratch. Unfortunately, their interfaces are of sufficient complexity that the use of more than one such library, or offering the user the choice of distribution library to be used, will almost certainly greatly complicate the parallel library code, particularly in the former case wherein the same data may be alternately or even simultaneously be 'managed' by more than one distribution library.

2 PADRE

The express purpose of PADRE is to provide comprehensive support for the distribution of data and the communication between different distributions of data, for both shared and distributed memory machines, and for both SPMD and MIMD execution models. While PADRE was not intended to be a mechanism for communication between separate applications (this type of functionality is the goal of other work [?]) there is at least one current effort to use it in support communication between disparate applications [?].

Three primary considerations have guided the design of PADRE. One is to provide functionality at a well-defined level of conceptual abstraction, in other words, to *not* attempt to be monolithic all-in-one data-distribution 'solution'. In practice PADRE operates on top of various lower-level libraries. The second is that PADRE be abstract with respect to several aspects of the user's (or more likely, a more specialized library's) data and the details of its distribution. The third is that PADRE provide high-level services without artificially constraining the user, that is, not restrict user access to the libraries on which PADRE depends, yet not require the user to work directly with, or even be aware of, these underlying libraries. In the same spirit, PADRE does not attempt to wrest control of the user's data—the user retains full control over the allocation and manipulation of the data

independently of PADRE.

Data distribution. PADRE is abstract with respect to several aspects of data distribution, notably the elementary type (e.g. int, float, or arbitarily complex structures), structure (e.g. arrays, particles), distribution (e.g. as elementary data elements, blocks or other structures of such elements), Object-oriented design, implementation using C++, and the use of the C++ class template mechanism gives the desired generality and performance.

Data distributions and alignment schemes are supported for both algorithm-based and table-based distributions—though the algorithm-based may be more efficient, table-based distributions permit a greater level of specification for support of sophisticated alignment of data. The implementation of distribution mechanisms is separate from PADRE itself; a key feature of PADRE is the ability to use multiple distribution mechanisms simultaneously with transparent translation between them. The choice of a particular distribution mechanism for a particular data set may be directly specified by the user, or determined by PADRE on the basis of user-supplied specifications or constraints on the data layout.

Current implementations of PADRE take advantage of existing specialized distribution libraries that are publicly available, for example, the Multiblock PARTI library from University of Maryland [?], the parallel gather-scatter library PGSLib from Cambridge Power Computing Associates [?], and KeLP from the San Diego Supercomputing Center [?]. PADRE is designed to make the addition of distribution libraries a minor task. Thus, more generally, we anticipate that PADRE will be used as a test-bed for evaluating new distribution schemes; alternation of data distribution libraries would be transparent to applications.

Translation between distribution mechanisms is achieved by identifying a partial ordering of the mechanisms and guaranteeing that every pair of mechanisms has a lower bound (that is, a distribution at least as general as either). This strategy allows the number of translations mechanisms to be linear, rather than quadratic, in the number of distribution mechanisms. More efficient and sophisticated mechanisms are planned for future work.

Communication. PADRE supports communication via *communication schedules* that are generated by the distribution library. Communication schedules may be executed externally to PADRE, typically by the runtime system. Besides modularizing the design, this allows the possibility of communications to be externally optimized with respect to relevant criteria such as the nature of a particular architecture or the number of communication events. PADRE is also independent of the underlying communication mechanism, e.g. MPI or PVM.

Schedules are provided for block-type transfers. Additional work supports unstructured-type communications as well; such unstructured communication support is a part of the parallel indirect addressing support required by the software frameworks. Unstructured communication within PADRE uses the PGSLib.

3 Overview of the design of PADRE

Even within PADRE distinct levels of conceptual abstraction are realized by a hierarchy of three principal classes. The need and justification for these particular abstractions is perhaps not self-evident but is born of considerable work in OVERTURE and POOMA

for which PADRE is the next evolutionary step. These three classes are briefly described following.

- **PADRE_Distribution.** The most abstract level is embodied by the PADRE_Distribution, which encodes such size-independent information as how the data may be decomposed, characteristics of ghost boundaries (local cached data), and axis-based descriptions for data domains that support them.
- **PADRE_Representation.** The intermediate level is embodied by the PADRE_Representation, which encodes the actual size of a data set, but is not associated with a particular data set. It also handles abstract communication schedules that are independent of a particular data set. A PADRE_Distribution may be shared by several PADRE_Representations.
- **PADRE_Descriptor.** The most concrete level is embodied by the PADRE_Descriptor, instances of which are in one-to-one correspondence with data sets. It handles, for example, concrete communication schedules. A PADRE_Representation may be shared by several PADRE_Descriptors.

The distribution libraries must provide an analogous three-level interface. In practice distribution libraries do not come so equipped; we have successfully provided them with appropriate 'wrappers'.

The principal PADRE classes are class templates parameterized by three user-defined classes, *UserDomain*, *UserLocalDescriptor*, and *UserCollection*. The UserDomain is effectively a description of the domain in which the data lives or may be broken up into. The UserLocalDescriptor contains a UserDomain object and a pointer to the user's data. It provides features needed for the user to exercise optional features of PADRE, for example, allocation of data by PADRE. The UserCollection is the user's class to be distributed, for example the prototypical array class.

4 PADRE in Practice

In managing the destribution of data PADRE has a special role in dynamic applications. Dynamic applications in a parallel environment require dynamic redistribution of data throughout the computation. One of the most complex examples of this is the use of adaptive mesh refinement (AMR). Within an AMR application refinement regions are added and deleted; through the process elements of the collection of grids (the adaptive grid) may be migrated or redistributed to maintain load balance. The additional complexity of parallel AMR applications lay in the interactions between the seperate grids of the adaptive grid. The dynamic behavior precludes the possibility of having convenient distributions that would simplify the operations between the many grids of an adaptive grid. Communication schedules between grids are thus complex and dynamic. Alignment between distributions are equally complex and must be available for efficient computations.

By sharing PADRE as a common manager for data distribution, OVERTURE and (eventually) POOMA may much more readily share significant subcomponents. For example, should an application using POOMA need a capability from OVERTURE such as AMR or parallel indirect addressing this may be provided with little difficulty.

The typical structure of a complete application using PADRE is given in Figure 1; in finer detail in Figure 2.

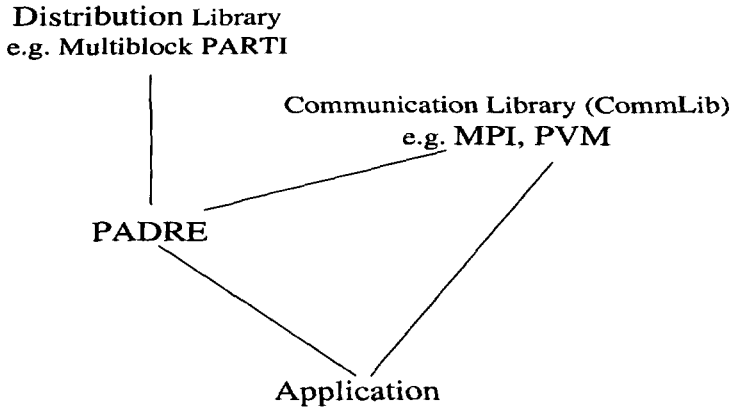

Figure 1: Gross structure of application using PADRE.

5 Results

PADRE has been successfully incorporated into A++/P++, using the Multiblock PARTI distribution library, with no apparent performance penalty. Currently other distribution libraries are being added and tested, and various optimizations, particularly with respect to data caching, have been implemented. In general, we expect no performance penalty from the use of PADRE over that of using any of the underlying distribution libraries.

6 Conclusion

Though PADRE supports only one aspect of the infrastructure for parallel libraries, it is intended to neatly and completely encapsulate a particularly complex part of that infrastructure. Other than defining an API it places no constraints on the application or framework using it. Since PADRE is a part of common infrastructure and leveraged by more than one parallel framework, it can provide greater flexibility in the distribution of data and in the use of that data (generation of complex communication schedules) than can be readily justified within a single parallel library or framework addressing a specific application area.

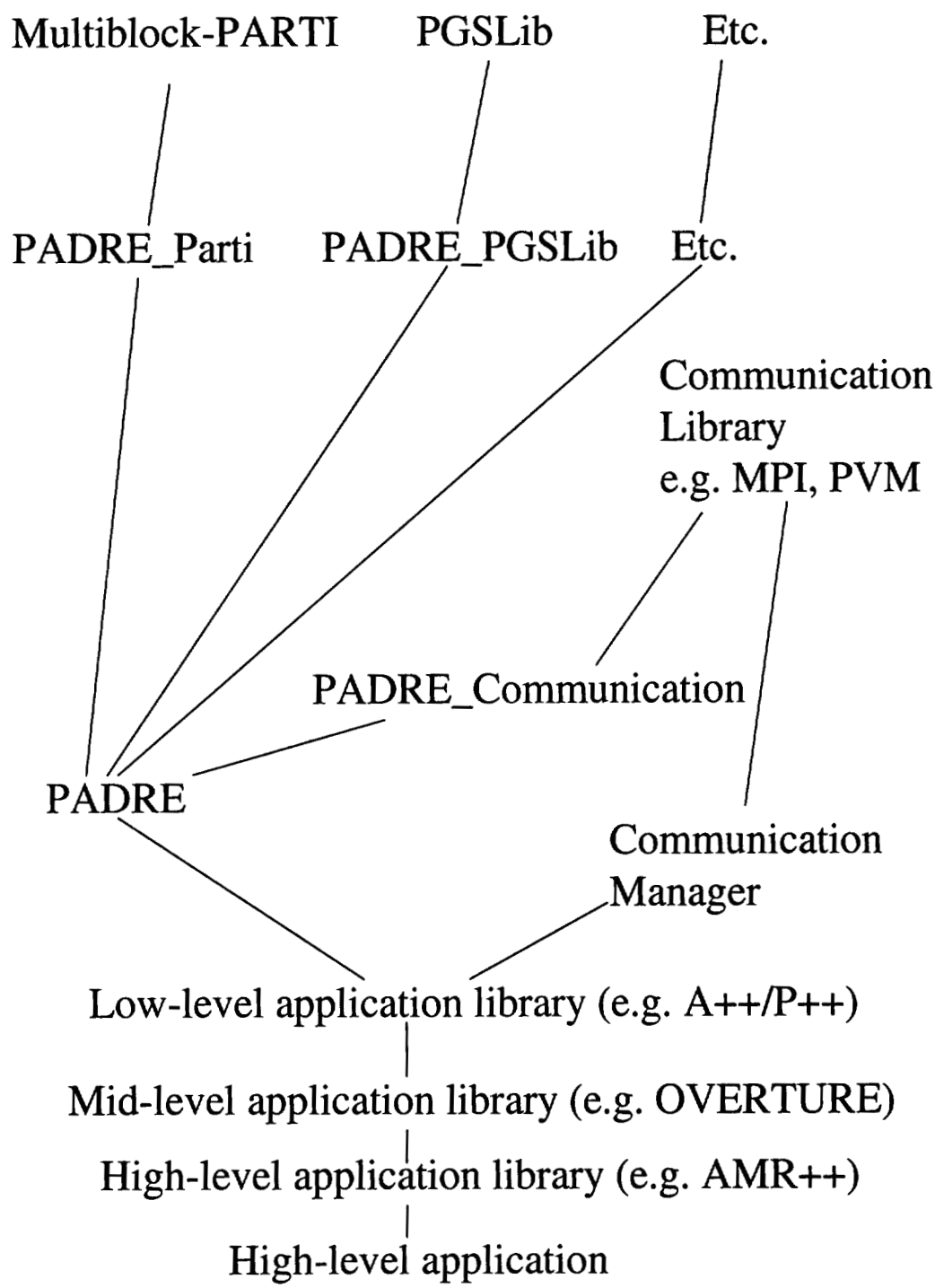

Figure 2: Fine structure of application using PADRE.